

# Clustering with Lattices in the Analysis of Graph Patterns

Edgar H. de Graaf, Joost N. Kok, and Walter A. Kusters

*Leiden Institute of Advanced Computer Science  
Leiden University, The Netherlands  
edegraaf@liacs.nl*

**Abstract.** Mining frequent subgraphs is an area of research where we have a given set of graphs (each graph can be seen as a transaction), and we search for (connected) subgraphs contained in many of these graphs. In this work we will discuss techniques used in our framework LATTICE2SAR for mining and analysing frequent subgraph data and their corresponding lattice information. Lattice information is provided by the graph mining algorithm GSPAN; it contains all supergraph-subgraph relations of the frequent subgraph patterns — and their supports.

LATTICE2SAR is in particular used in the analysis of frequent graph patterns where the graphs are molecules and the frequent subgraphs are fragments. In the analysis of fragments one is interested in the molecules where patterns occur. This data can be very extensive and in this paper we focus on a technique of making it better available by using the lattice information in our clustering. Now we can reduce the number of times the highly compressed occurrence data needs to be accessed by the user. The user does not have to browse all the occurrence data in search of patterns occurring in the same molecules. Instead one can directly see which frequent subgraphs are of interest.

## 1 Introduction

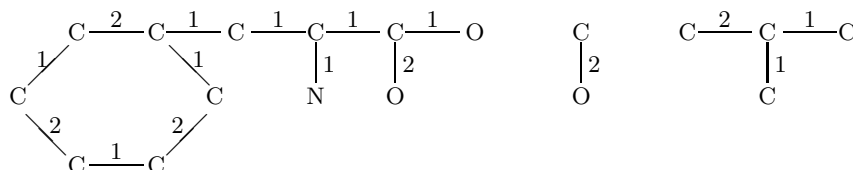
Mining frequent patterns is an important area of data mining where we discover substructures that occur often in (semi-)structured data. The research in this work will be in the area of frequent subgraph mining. These *frequent subgraphs* are connected vertex- and edge-labeled graphs that are subgraphs of a given set of graphs, traditionally also referred to as *transactions*, at least *minsupp* times. The example of Figure 1 shows a graph and two of its subgraphs.

In this paper we will use results from frequent subgraph mining and visualize the frequent subgraphs by means of clustering, where their co-occurrences in the same transactions are used in the distance measure. Clustering makes it possible to obtain a quicker selection of the right frequent subgraphs for a more detailed look at their occurrence.

Before explaining what is meant by lattice information we first need to discuss *child-parent* relations in frequent subgraphs, also known as patterns. Patterns are generated by extending smaller patterns with one extra edge. The smaller pattern can be called a *parent* of the bigger pattern that it is extended to. If we

would draw all these relations, the drawing would be shaped like a lattice, hence we call this data *lattice information*.

We further analyse frequent subgraphs and their corresponding lattice information with different techniques in our framework LATTICE2SAR for mining and analysing frequent subgraph data. One of the techniques in this framework is the analysis of graphs in which frequent subgraphs occur, via clustering. Another important functionality is the browsing of lattice information from parent to child and from child to parent. In this paper we will present the clustering.



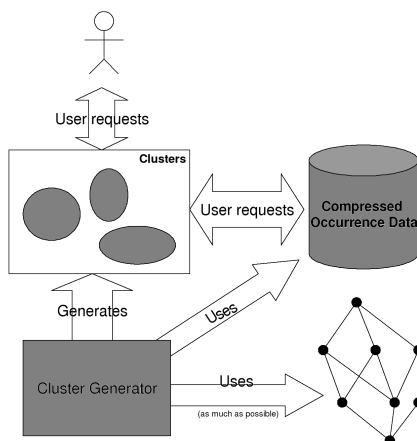
**Fig. 1.** An example of a possible graph (the amino acid Phenylalanine) in the molecule dataset and two of its many (connected) subgraphs, also called patterns or fragments.

Our working example is the analysis of patterns (fragments) in molecule data, since LATTICE2SAR was originally made to handle molecule data. Obviously molecules are stored in the form of graphs, the molecules are the transactions (see Figure 1 for an example). However, the techniques presented here are not particular to molecule data (we will also not discuss any chemical or biological issues). For example one can extract user behaviour from access logs of a website. This behaviour can be stored in the form of graphs and can as such be mined with the techniques presented here.

The distance between patterns can be measured by calculating in how many graphs (or molecules) only one of the two patterns occurs. If this never happens then these patterns are very close to each other. If this always happens then their distance is very large. In both cases the user is interested to know the reason. In our working example the chemist might want to know which different patterns seem to occur in the same subgroup of effective medicines or which patterns occur in different subgroups of effective medicines. In this paper we will present an approach to solve this problem that uses clustering. Furthermore all occurrences for the frequent subgraphs will be discovered by a graph mining algorithm and this occurrence information will be highly compressed before storage. Because of this, requesting these occurrences will be costly. Through our method of clustering time will be saved if the user uses the clusters to select interesting patterns, see Figure 2 for an overview.

We will define our method of clustering and show its usefulness. To this end, this paper makes the following contributions:

- Our first contribution will be that we will **introduce an algorithm for clustering frequent subgraphs** allowing the user to quickly see interesting relations, e.g., subgraphs occur in the same transactions, and quicker select the right occurrence details from the compressed storage (all sections and specifically Section 4).



**Fig. 2.** Diagram of how the cluster browsing part of LATTICE2SAR is used.

- We will define a measure of calculating distances between patterns and **show how the lattice information can be used for faster calculation** (Section 2 and Section 3).
- The lattice information can be used to make groups of patterns and in this way **clarity of the visualization can be improved** due to less points in the 2-dimensional model. In Section 3 we will introduce this preprocessing step.
- Preprocessing will also make **faster clustering possible by reducing cluster points**, diminishing requests for occurrence counting (Section 3 and Section 5).
- Finally through experiments **the effectiveness of our clustering is shown** and the resulting cluster model is analyzed (Section 5).

This research is related to research on clustering, in particular of molecules. Also our work is related to frequent subgraph mining and frequent pattern mining when lattices are discussed. In [10] Zaki et al. discuss different ways for searching through the lattice and they propose the ECLAT algorithm.

Clustering in the area of biology is important because of the visualization that it can provide. E.g., [7] Samsonova et al. discuss the use of Self-Organizing Maps (SOMs) for clustering protein data. In general our work is related to SOMs as developed by Kohonen (see [3]), in the sense that SOMs are also used to visualize data through a distance measure. SOMs have been used in a biological context many times, for example in [2, 6]. In some cases molecules are clustered via numeric data describing each molecule, in [8] clustering such data is investigated.

Our package of mining techniques for molecules is called LATTICE2SAR; it makes use of a graph miner called GSPAN, introduced in [9] by Yan and Han. This implementation generates the patterns organized as a lattice and a separate compressed file of occurrences of the patterns in the graph set (molecules).

In this work a method of pushing and pulling points in accordance with a distance measure is used. This technique was used before by Cocx et al. in [1] to cluster criminal careers and was developed in [4]. This method of clustering was

chosen since we only know the distance between two patterns. We don't know the precise  $x$  and  $y$  coordinates of the patterns, so we can not use standard methods of discovering clusters, e.g., K-Means (see [5]). The algorithm from [4] is different from clustering with self-organizing maps as SOMs adapt the weight vector of each neuron toward an input vector. In our problem no such input vector exists and each point in the cluster is linked with one graph or one group of graphs.

## 2 Distance Measure

For any clustering algorithm you have to at least know the distance between the points in the model. As was mentioned in the introduction, we are interested to know if patterns occur in the same graphs in the dataset of graphs. Patterns in this work are *connected frequent subgraphs* where all vertices of the subgraph can be found in *minsupp* graphs of the dataset with matching labels and connections between the vertices (see Figure 1 for an example). If a subgraph occurs at different positions in a graph, it is counted only once. Here *minsupp* is a user-defined threshold for frequency.

The distance measure will compute how often frequent subgraphs occur in the same graphs of the dataset. In the case of our working example it will show if different fragments (frequent subgraphs) exist in the same molecules. Formally we will define the distance measure in the following way (for graphs  $g_1$  and  $g_2$ ):

$$dist(g_1, g_2) = \frac{support(g_1) + support(g_2) - 2 \cdot support(g_1 \wedge g_2)}{support(g_1 \vee g_2)} \quad (1)$$

Here  $support(g)$  is the number of times a (sub)graph  $g$  occurs in the set of graphs;  $support(g_1 \wedge g_2)$  gives the number of graphs (or transactions) with both subgraphs and  $support(g_1 \vee g_2)$  gives the number of graphs with at least one of these subgraphs. The numerator of the *dist* measure computes the number of times the two graphs do not occur together in one graph of the dataset. We divide by  $support(g_1 \vee g_2)$  to make the distance independent from the total occurrence, thereby normalizing it. We can reformulate *dist* in the following manner:

$$dist(g_1, g_2) = \frac{support(g_1) + support(g_2) - 2 \cdot support(g_1 \wedge g_2)}{support(g_1) + support(g_2) - support(g_1 \wedge g_2)} \quad (2)$$

In this way we do not need to separately compute  $support(g_1 \vee g_2)$  by counting the number of times subgraphs occur in the graphs in the dataset, saving us the time needed to access this compressed dataset.

*Example 1.* Say fragment  $A$  has a string 11100011 indicating in which molecule it occurs. So, from left to right, fragment  $A$  occurs in the first 3 molecules and in the last 2, where a 1 indicates that a fragment occurs and a 0 indicates a non-occurrence. If fragment  $B$  has string 01111000, we can see that either fragment  $A$  or  $B$  or both ( $support(A \vee B)$ ) occur in 7 molecules ( $5 + 4 - 2$ ). Similarly,  $support(A \wedge B) = 2$ , and  $dist(A, B) = 5/7$ .  $\square$

The distance measure satisfies the usual requirements, such as the triangular inequality. Note that  $0 \leq \text{dist}(g_1, g_2) \leq 1$  and  $\text{dist}(g_1, g_2) = 1 \Leftrightarrow \text{support}(g_1 \wedge g_2) = 0$ , so  $g_1$  and  $g_2$  have no common transactions in this case. If  $\text{dist}(g_1, g_2) = 0$ , both subgraphs occur in the same transactions, but are not necessarily equal.

The distances are computed *after* discovering all frequent subgraphs with GSPAN. Obviously, while computing the support for the graphs not all frequent subgraphs are known and not all distances can be computed while running GSPAN.

### 3 Preprocessing: Grouping

Possibly we will discover many frequent subgraphs, depending on the chosen minimal support. Next to having to store the supports of all frequent subgraphs, we will also have to store the distance for all frequent subgraph combinations in order to decide clusters fast. If we have  $n$  frequent subgraphs then storing the support for all  $n(n-1)/2$  combinations might be too much. However many frequent subgraphs often are very similar in both structure and support. Furthermore, for these very similar frequent subgraphs there often exists a parent-child relation.

Now we will propose a preprocessing step where we first group close subgraphs and we will treat them as one point in our cluster model. This will reduce the number of points in the cluster model and the number of distances that have to be stored and/or decided. This will not only speed up the process of deciding the distance between groups or graphs, another benefit is that the overview in the visualization will be improved because less of the same graphs are in the 2-dimensional cluster model. Furthermore it will reduce exploration time for the expert, because many of these redundant graphs are grouped. If an expert wants to view the occurrence of a graph then (s)he can select just one of the group to be retrieved from the compressed set, since their occurrence is almost equal.

The formula for the distance between supergraph  $g_2$  and subgraph  $g_1$  originates from Equation 2, where  $\text{support}(g_1 \wedge g_2) = \text{support}(g_2)$ :

$$\begin{aligned} \text{dist}(g_1, g_2) &= \frac{\text{support}(g_1) + \text{support}(g_2) - 2 \cdot \text{support}(g_2)}{\text{support}(g_1) + \text{support}(g_2) - \text{support}(g_2)} \\ &= \frac{\text{support}(g_1) - \text{support}(g_2)}{\text{support}(g_1)} \end{aligned}$$

*Example 2.* If we take a fragment  $A$  with occurrence string 11110011 and a fragment  $B$  (where  $A$  is a subgraph of  $B$ ) with string 11110000, then we of course see that fragment  $A$  occurs at least in all molecules where  $B$  occurs. In this case  $\text{support}(A \wedge B) = \text{support}(B) = 4$ .  $\square$

All information used to compute these distances can be retrieved from the lattice information provided by the graph mining algorithm, when we focus on the subgraph-supergraph pairs. This information is needed by the graph mining

algorithm to discover the frequent subgraphs and so the only extra calculating is done when *dist* does a search in this information.

Of course, many graphs have no parent-child relation and for this reason we define *pregroup\_dist* in the following way:

$$\text{pregroup\_dist}(g_1, g_2) = \begin{cases} \text{dist}(g_1, g_2) & \text{if } g_2 \text{ is a supergraph of } g_1 \\ & \text{or } g_1 \text{ is a supergraph of } g_2 \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

Note that *pregroup\_dist*( $g_1, g_2$ ) < 1 if  $g_1$  is a subgraph of  $g_2$  and has non-zero support, or the other way around.

Now we propose the PREGROUP algorithm that will organize close subgraphs/supergraphs into groups. The algorithm is based on hierarchical clustering and because of this we need to define how we decide the distance between clusters  $C_1 = \{g_1, g_2, \dots, g_n\}$  and  $C_2 = \{h_1, h_2, \dots, h_m\}$ :

$$\text{cluster\_dist}(C_1, C_2) = \begin{cases} \max(PG) & \text{if } PG \neq \emptyset \\ -1 & \text{otherwise} \end{cases} \quad (4)$$

$$PG = \{\text{pregroup\_dist}(g, h) \mid g \in C_1, h \in C_2, \text{pregroup\_dist}(g, h) \neq 1\}$$

Two clusters should not be merged if their graphs do not have a supergraph-subgraph relation, so we do not consider graphs where *pregroup\_dist*( $g, h$ ) = 1. The value of *cluster\_dist* is -1 if no maximal distance exists, and clusters will not be merged in the algorithm.

The outline of the algorithm is the following:

---

```

initialize  $\mathcal{P}$  with sets of subgraphs of size 1 from the lattice
while  $\mathcal{P}$  was changed or was initialized
    Select  $C_1$  and  $C_2$  from  $\mathcal{P}$  with minimal cluster_dist ( $C_1, C_2$ )  $\geq 0$ 
    if cluster_dist( $C_1, C_2$ )  $\leq \text{maxdist}$  then
         $\mathcal{P} = \mathcal{P} \cup \{C_1 \cup C_2\}$ 
        Remove  $C_1$  and  $C_2$  from  $\mathcal{P}$ 

```

---

PREGROUP

---

The parameter *maxdist* is a user-defined threshold giving the largest distance allowed for two clusters to be joined.

## 4 Relative Positioning of Groups

The information we need to store concerning the occurrence of subgraph patterns can be huge. However, in some cases the user might want to have this information, e.g., in our working example the scientist might want to closer investigate molecules (transactions) contain a specific pattern.

Interesting information for any user is to see how often the groups (clusters) of subgraphs occur in the same transactions (graphs) within the dataset.

Here we will visualize this co-occurrence by positioning all groups randomly in a 2-dimensional area and adapting their position a number of times with the formulas from [4]. In our model for  $eucl\_dist(C_1, C_2)$  we take the Euclidean distance between the 2-dimensional coordinates of the points corresponding with the two groups (of frequent subgraphs)  $C_1$  and  $C_2$ .

The graphs in a group occur in almost all the same transactions, depending on the chosen  $maxdist$ . So, in order to speed up clustering, the distance between groups is assumed to be the distance between any of the points of the two groups. In our algorithm we choose to define the distance between groups as the distance between a smallest graph of each of the two groups ( $size$  gives the number of vertices): for  $g_1 \in C_1$  and  $g_2 \in C_2$  with  $size(g_1) = \min(\{size(g) \mid g \in C_1\})$  and  $size(g_2) = \min(\{size(g) \mid g \in C_2\})$ , we let  $group\_dist(C_1, C_2) = dist(g_1, g_2)$ .

The coordinates  $(x_{C_1}, y_{C_1})$  and  $(x_{C_2}, y_{C_2})$  of the points corresponding with  $C_1$  and  $C_2$  are adapted by applying the following formulas:

1.  $x_{C_1} \leftarrow x_{C_1} - \alpha \cdot (eucl\_dist(C_1, C_2) - group\_dist(C_1, C_2)) \cdot (x_{C_1} - x_{C_2})$
2.  $y_{C_1} \leftarrow y_{C_1} - \alpha \cdot (eucl\_dist(C_1, C_2) - group\_dist(C_1, C_2)) \cdot (y_{C_1} - y_{C_2})$
3.  $x_{C_2} \leftarrow x_{C_2} + \alpha \cdot (eucl\_dist(C_1, C_2) - group\_dist(C_1, C_2)) \cdot (x_{C_1} - x_{C_2})$
4.  $y_{C_2} \leftarrow y_{C_2} + \alpha \cdot (eucl\_dist(C_1, C_2) - group\_dist(C_1, C_2)) \cdot (y_{C_1} - y_{C_2})$

Here  $\alpha$  ( $0 \leq \alpha \leq 1$ ) is the user-defined learning rate.

Starting with random coordinates for the groups, we will build a 2-dimensional model of relative positions between groups by randomly choosing two groups  $r$  times and applying the formulas.

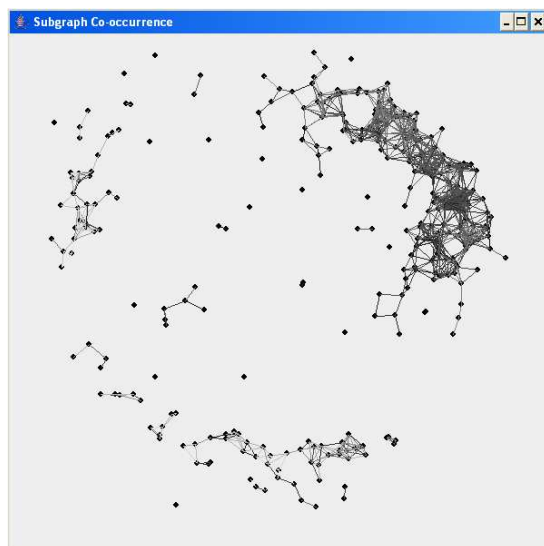
## 5 Results and Performance

The experiments are organized such that we first show the cluster model to approximate the distances correctly. Secondly through experiments we show the speed-up due to making groups first. We make use of a dataset, the *molecule dataset*, containing 4,069 molecules; from this we extracted a lattice containing the 1,229 most frequent subgraphs.

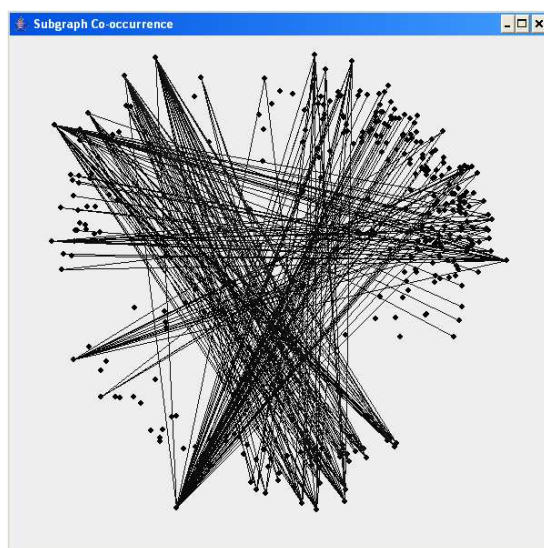
All experiments were performed on an Intel Pentium 4 64-bits 3.2 GHz machine with 3 GB memory. As operating system Debian Linux 64-bits was used with kernel 2.6.8-12-em64t-p4.

Figure 3 shows how points, that represent subgraphs occurring in the same graphs (molecules) of the dataset, cluster together. We made lines between points if their Euclidean distance is  $\leq 0.05$ . The darker these lines the lower their actual distance and in this way one can see gray clusters of close groups of subgraphs. Some groups are placed close but their actual distance is not close (they are light grey). This is probably caused by the fact that these groups do not occur together with some specific other groups, so being far away from these other ones.

In Figure 4 we make lines between points with a Euclidean distance  $\geq 0.95$ . The darker these lines the higher their actual distance. The figure shows their actual distance to be big also (the lines are black). Also Figure 4 shows bundles



**Fig. 3.** Clusters for graphs in the molecule dataset built in 24.5 seconds, connecting points at distance 0.05 or lower ( $\alpha = 0.1$ ,  $maxdist = 0.1$ ,  $r = 1,000,000$ ).

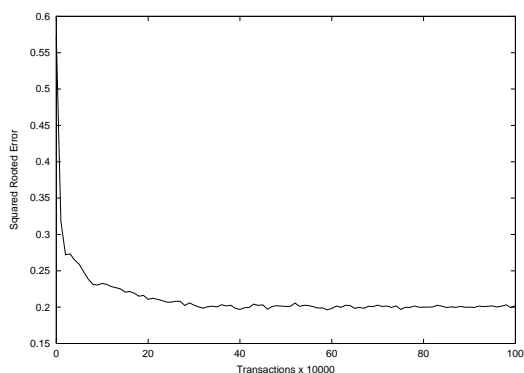


**Fig. 4.** Clusters of graphs in the molecule dataset built in 24.5 seconds, connecting points at distance 0.95 or higher ( $\alpha = 0.1$ ,  $maxdist = 0.1$ ,  $r = 1,000,000$ ).

of lines going to one place. This probably is again caused by groups not occurring together with the same other groups.

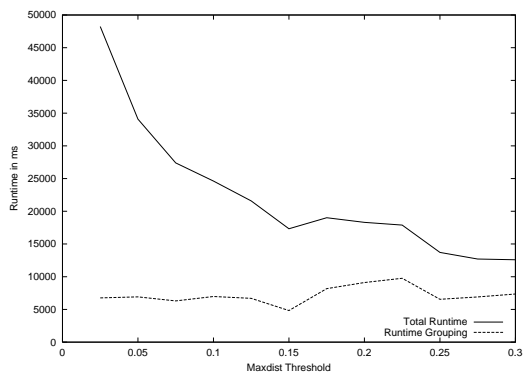
The error for the cluster model decreases quickly, see Figure 5. At some point it becomes very hard to reduce the error further.





**Fig. 5.** Root squared error for distance given by the cluster model ( $\alpha = 0.1$ ).

In one experiment we assumed that the distances could not be stored in memory. In this experiment we first clustered 1,229 patterns without grouping, taking 81 seconds. However, grouping reduced the number of requests to the compressed occurrence data and because of this with grouping clustering took 48 seconds ( $\alpha = 0.1$ ,  $r = 1,000,000$ ,  $maxdist = 0.1$ ).



**Fig. 6.** Average runtime for the molecule dataset with varying  $maxdist$  ( $\alpha = 0.1$ ,  $r = 1,000,000$ ).

Our final experiment was done to show how the runtime is influenced by the  $maxdist$  threshold and how much the preprocessing step influences runtime. Here we assume the distances between clusters can be stored in memory. In Figure 6 the influence on runtime is shown. The time for preprocessing appears to be more or less stable, but the total runtime drops significantly.

## 6 Conclusions and Future Work

Presenting data mining results to the user in an efficient way is important. In this paper we propose a preprocessing step for an existing method of clustering and we apply this method to frequent subgraphs, which was not done before.

The model can be built faster with the clustering algorithm because of the grouping of the subgraphs, the preprocessing step. The groups also remove redundant points from the visualization that represents very similar subgraph patterns. Finally the model enables the user to quickly select the right subgraphs for which the user wants to investigate the graphs (or molecules) in which the frequent subgraphs occur.

In the future we want to take a closer look at grouping where the types of vertices and edges and their corresponding weight also decide their group. Furthermore, we want to investigate how we can compress occurrence more efficiently and access it faster.

**Acknowledgments** This research is carried out within the Netherlands Organization for Scientific Research (NWO) MISTA Project (grant no. 612.066.304). We thank Siegfried Nijssen for his implementation of GSPAN.

## References

1. Bruin, J.S. de, Cocx, T.K., Kusters, W.A., Laros, J.F.J. and Kok, J.N.: *Data Mining Approaches to Criminal Career Analysis*, in Proc. 6th IEEE International Conference on Data Mining (ICDM 2006), pp. 171–177.
2. Hanke, J., Beckmann, G., Bork, P. and Reich, J.G.: *Self-Organizing Hierarchic Networks for Pattern Recognition in Protein Sequence*, Protein Science Journal 5 (1996), pp. 72–82.
3. Kohonen, T.: *Self Organizing Maps*, Volume 30 of Springer Series in Information Science, Springer, second edition, 1997.
4. Kusters, W.A. and Wezel, M.C. van: *Competitive Neural Networks for Customer Choice Models*, in E-Commerce and Intelligent Methods, volume 105 of Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer, 2002, pp. 41–60.
5. MacQueen, J.B.: *Some Methods for Classification and Analysis of Multivariate Observations*, in Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.
6. Mahony, S., Hendrix, D., Smith, T.J. and Golden, A.: *Self-Organizing Maps of Position Weight Matrices for Motif Discovery in Biological Sequences*, Artificial Intelligence Review Journal 24 (2005), pp. 397–413.
7. Samsonova, E.V., Bäck, T., Kok, J.N. and IJzerman, A.P.: *Reliable Hierarchical Clustering with the Self-Organizing Map*, in Proc. 6th International Symposium on Intelligent Data Analysis (IDA 2005), pp. 385–396.
8. Xu, J., Zhang, Q. and Shih, C.-K.: *V-Cluster Algorithm: A New Algorithm for Clustering Molecules Based Upon Numeric Data*, Molecular Diversity 10 (2006), pp. 463–478.
9. Yan, X. and Han, J.: *gSpan: Graph-Based Substructure Pattern Mining*. In Proc. 2002 IEEE International Conference on Data Mining (ICDM 2002), pp. 721–724.
10. Zaki, M., Parthasarathy, S., Ogihara, M. and Li, W.: *New Algorithms for Fast Discovery of Association Rules*, in Proc. 3rd International Conference on Knowledge Discovery and Data Mining (KDD 1997), pp. 283–296.